

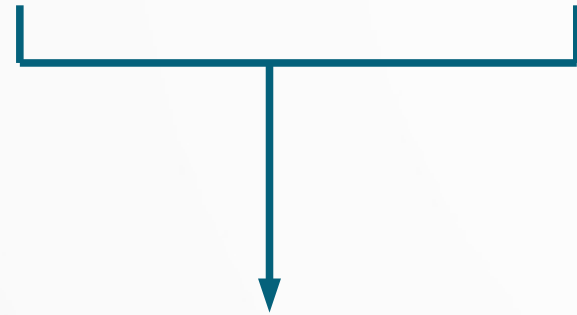


# Generative Adversarial Malware Classifier Deceptor with Reinforcement Learning

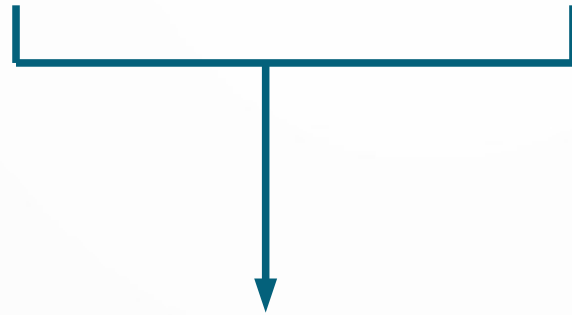
**Nancirose Piazza**

# Definitions

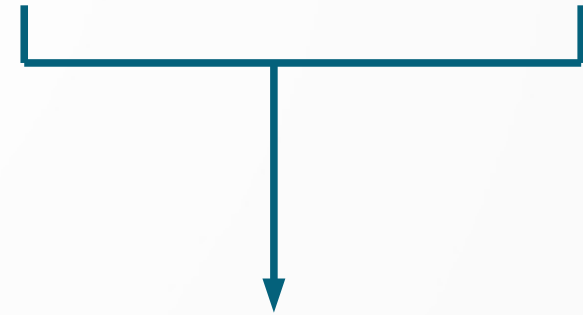
Generative Adversarial Malware Classifier Deceptor with Reinforcement Learning



Generate modified  
Malware that...



Can bypass a Malware  
Classification as  
Benign...



Through an  
unsupervised  
Machine learning  
method

# Reinforcement Learning Fundamentals

**Action Space:** the full set of possible actions.

**Agent:** The thing that takes actions.

**State:** An agent's immediate condition.

**Environment:** Takes in Agent and Agent's state. Then, outputs an observation.

**Episode:** Steps which is how many times the model is updated from each taken action before the terminal state.

**Observation:** A reward and next Agent's state.

**Reward:** Performance feedback from an instance of the environment.

**Policy:** How the agent decides the next action.

An action: `rename_section`

Environment: Mutating sample, vectorizing and testing against the classifier.

An Episode: Send (`rename_section`, a malicious sample) to environment. New score below threshold, environment returned a value of .80 which is below threshold .9.

Observation: Sample evaded, end episode, give big reward.

Reward: 10

State: Evaded.

Policy: Boltzmann Exploration Policy.

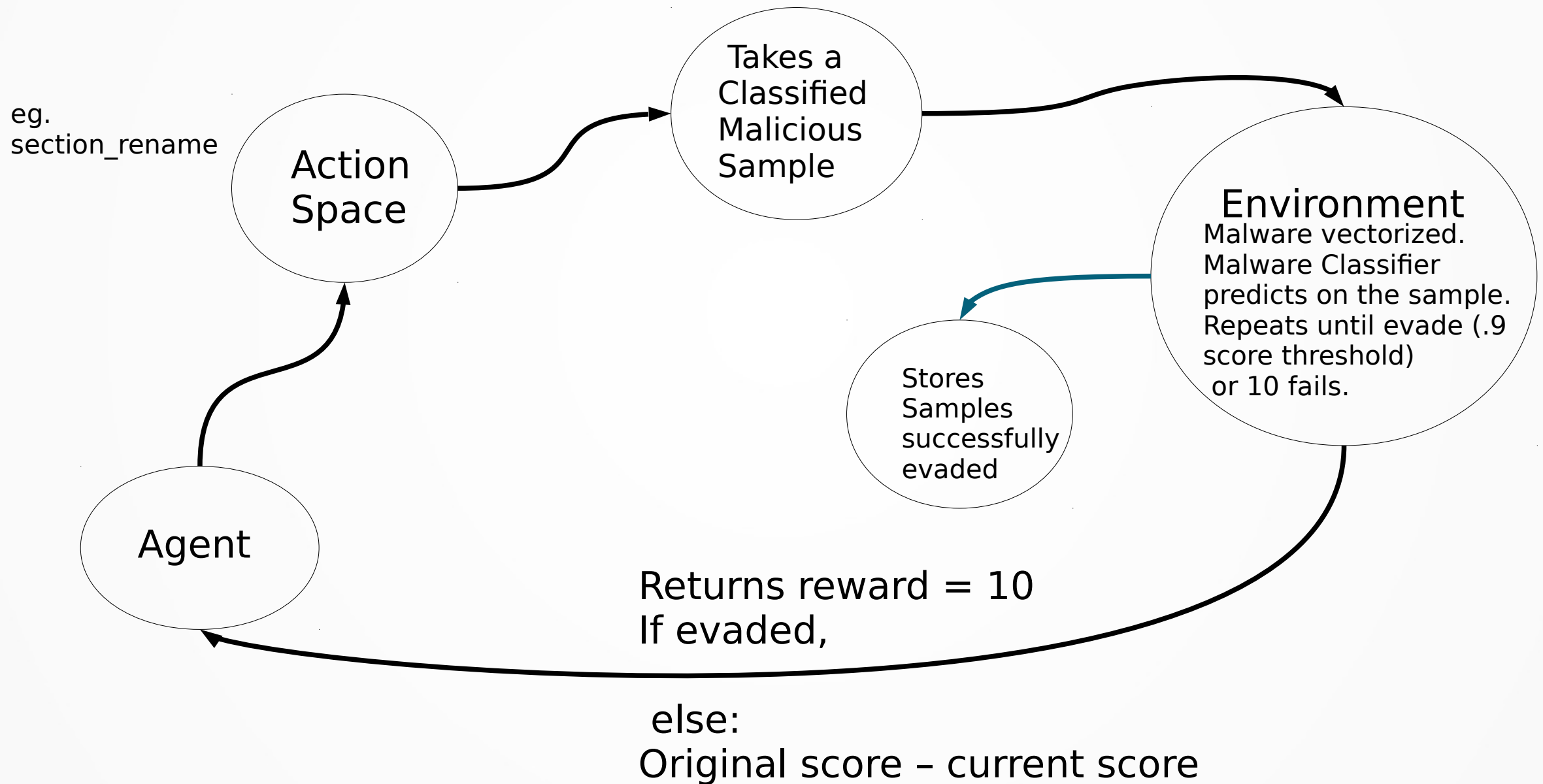
# Reinforcement Learning Continued.

- Model-based
  - Markov Decision Process
  - All elements are known:
    - $S \rightarrow$  State
    - $A \rightarrow$  Set of Actions from Action Space
    - $R(s,a) \rightarrow$  The function that returns reward for action  $a$ .
    - $T(s' | s,a) \rightarrow$  The transition probability function for environment transition to state  $s'$  given state  $s$  and action  $a$ .
    - A model of the environment is made where probabilities of all states and transitional-state from all possible actions are calculated.
- Model-Free
  - Does not use the function  $T$  nor function  $R$ .
    - “Trial and Error”
    - No model of the environment, no prior knowledge and learns from experience.

# RL continued. Again

- Temporal Difference (TD) Learning
  - Model-free method
  - Predictions are adjusted before final outcome unlike Monte Carlo methods.
- To calculate an optimal policy:
  - Q-Learning (Quality)
    - Method of finding an optimal policy that maximizes total reward (expected reward) over all steps up until current state. Uses Q-table which is a matrix that assigns pairs of actions with states with an overall expected reward value.
    - Off-policy – unlike on-policy, does not use its current policy to update Q-values.
      - Deep QL
        - Instead of a paired action-state Q-table, a neural network layer is used where the input layer is the states and the output layer is the actions.
      - Double DQL (Double Deep Q-Learning)
        - Q and Q' are networks with identical architecture, To update  $Q(s,a)$ , the action is selected from Q with maximum expected reward given state s, but Q' is given the state and action to calculate the Q-value.

# Generative Adversarial Malware Classifier Deceptor



# Model Architecture

- DQL Agent: Double DQL Architecture, making it off-policy.
- Policy: Boltzmann Exploration Policy – stochastic policy where an action is selected by a generated probability of the softmax expected rewards/Q-values.
- Memory : Each malware sample is an experience
- Optimizer: RMSProp
- Metric: mae
- Iterations = Number of Training samples (Episode) \* 10,000 \* 5  
= 1.5 days of training

Action  
space size →

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 2350)	0
dropout_1 (Dropout)	(None, 2350)	0
dense_1 (Dense)	(None, 1024)	2407424
batch_normalization_1 (Batch Normalization)	(None, 1024)	4096
elu_1 (ELU)	(None, 1024)	0
dense_2 (Dense)	(None, 256)	262400
batch_normalization_2 (Batch Normalization)	(None, 256)	1024
elu_2 (ELU)	(None, 256)	0
dense_3 (Dense)	(None, 11)	2827
activation_1 (Activation)	(None, 11)	0
Total params: 2,677,771		
Trainable params: 2,675,211		
Non-trainable params: 2,560		
None		



# Before the Results

- Boltzmann Q Exploration Policy makes it a stochastic model. The results can vary.
- The random selection for Benign section names is also a distribution that can contribute to variation in results.
- The Malware Classifier is a Gradient Boost Decision Trees Model
- The original dataset size was 14. The test split is set to .2-- which allows up to 2 samples in test set. However, only one sample classified as Malicious from the Malware Classifier. Only 5 out of the 14 samples were classified as Malicious from the Classifier.



# Ransomwares

Able to Evade with policy.

Classified as Benign

- ('Ransomware.Satana\_sha256', '683a09da219918258c58a7f61f7dc4161a3a7a377cf82a31b840baabfb9a4a96')
- ('Ransomware.Petrwrap\_sha256', 'd975bc89371be98a80f9979e4a3c517590029e61ff36605b48b883c723024bde')
- ('Ransomware.Petrwrap\_sha256', '4340f4c633a53d45de440293d1f09f839467cc4734f499b466e9c58ee7493020')
- ('Ransomware.Petrwrap\_sha256', '3419e0d470f83569be0927128b3e5f992800ceb8f9019fc44763876ed6d8000c')
- ('Ransomware.Petrwrap\_sha256', '027cc450ef5f8c5f653329641ec1fed91f694e0d229928963b30f6b0d7d3a745')
- ('Ransomware.Petrwrap\_sha256', 'e5c643f1d8ecc0fd739d0bbe4a1c6c7de2601d86ab0fff74fd89c40908654be5')
- ('Ransomware.Radamant\_sha256', '2c4c8066a1a7dfdf42c57ff4f9016f1ba05bcb004ff8b0ffc0989165d2ad30e2')
- ('Ransomware.Petya\_sha256', '4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1ea45308b8c49b950655c')
- ('Ransomware.Petya\_sha256', '26b4699a7b9eeb16e76305d843d4ab05e94d43f3201436927e13b3ebafa90739')
- ('Ransomware.Unnamed\_0\_shasum', '517ac5506a5488a1193686f66cb57ad3288c2258c510004edb2f361b674526cc')
- ('Ransomware.Cryptowall\_sha256', '45317968759d3e37282ceb75149f627d648534c5b4685f6da3966d8f6fca662d')
- ('Ransomware.Cerber\_sha256', 'e67834d1e8b38ec5864cfa101b140aeaba8f1900a6e269e6a94c90fcbfe56678')
- ('Ransomware.Mamba\_sha256', '2ecc525177ed52c74ddaaacd47ad513450e85c01f2616bf179be5b576164bf63')
- ('Ransomware.WannaCry\_sha256', 'ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa')

Evaded Randomly

# Results on Test Set

```
Successful (random):  
(2c4c8066a1a7dfdf42c57ff4f9016f1ba05bcb004ff8b0ffc0989165d2ad30e2, ['upx_unpack'])  
  
Successful (score):  
None
```

6x iterated on the test set. This was the generic output.

```
Success rate (random chance): 1.0  
That is: 1 out of 1  
Success rate (dqn_score): 0.0  
That is: 0 out of 1
```

Though, the first model was able to have some instances correct, after training the model more, it moved away from that local minimum.

# Result on Full Set

Success rate (random chance): 0.4

That is: 2 out of 5

Success rate (dqn\_score): 0.0

That is: 0 out of 5

Success rate (random chance): 0.2

That is: 1 out of 5

Success rate (dqn\_score): 0.2

That is: 1 out of 5

Success rate (random chance): 0.6

That is: 3 out of 5

Success rate (dqn\_score): 0.2

That is: 1 out of 5

Successful (score):  
None

Successful (score):  
(e5c643f1d8ecc0fd739d0bbe4a1c6c7de2601d86ab0fff7  
4fd89c40908654be5,['section\_rename'])

Successful (score):  
(517ac5506a5488a1193686f66cb57ad3288c2258c5100  
04edb2f361b674526cc,['section\_rename',  
'section\_rename', 'section\_rename',  
'section\_rename', 'section\_rename', 'section\_rename',  
'section\_rename', 'section\_rename'])

# Results on Train

Success rate (random chance): 0.0

That is: 0 out of 4

Success rate (dqn\_score): 0.25

That is: 1 out of 4

Success rate (random chance): 0.25

That is: 1 out of 4

Success rate (dqn\_score): 0.0

That is: 0 out of 4

Success rate (random chance): 0.25

That is: 1 out of 4

Success rate (dqn\_score): 0.5

That is: 2 out of 4

Successful (score):

(517ac5506a5488a1193686f66cb57ad3288c2258c51000  
4edb2f361b674526cc,['section\_rename',  
'section\_rename', 'section\_rename',  
'section\_rename', 'section\_rename', 'section\_rename',  
'section\_rename'])

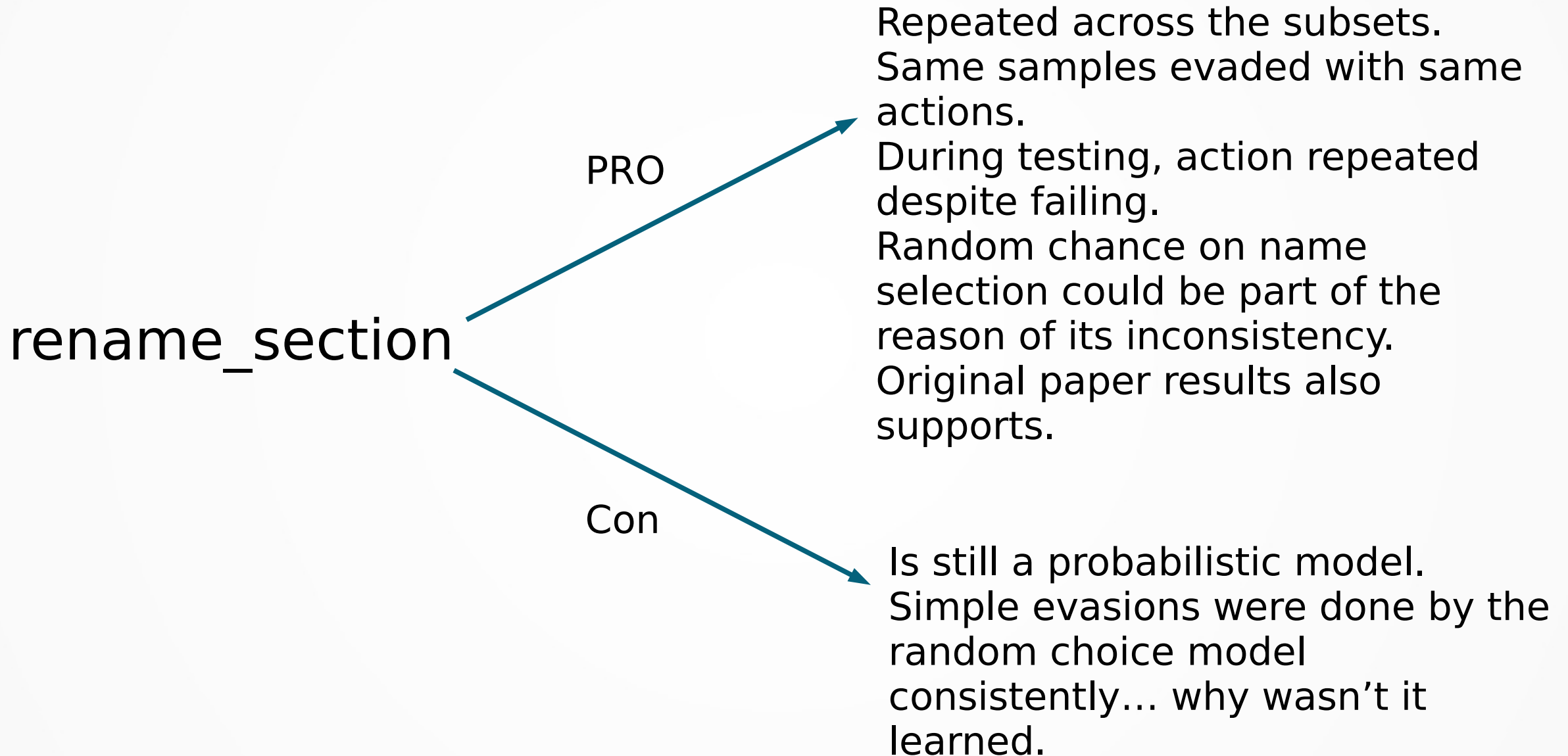
Successful (score):

None

Successful (score):

(4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1ea4530  
8b8c49b950655c,['section\_rename', 'section\_rename',  
'section\_rename', 'section\_rename', 'section\_rename'])  
(e5c643f1d8ecc0fd739d0bbe4a1c6c7de2601d86ab0fff74fd89  
c40908654be5,['section\_rename'])

# Model Learned or Luck?



# What can be done?

- It was trained on a stochastic policy, would a deterministic policy provide different test results?  
Let's see... On full set

```
Success rate (random chance): 0.4  
That is: 2 out of 5  
Success rate (dqn_score): 0.4  
That is: 2 out of 5
```

```
Success rate (random chance): 0.2  
That is: 1 out of 5  
Success rate (dqn_score): 0.0  
That is: 0 out of 5
```

```
Success rate (random chance): 0.4  
That is: 2 out of 5  
Success rate (dqn_score): 0.2  
That is: 1 out of 5
```

```
Successful (score):  
(e5c643f1d8ecc0fd739d0bbe4a1c6c7de2601d86ab0fff7  
4fd89c40908654be5,['section_rename',  
'section_rename', 'section_rename'])  
(4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1e  
a45308b8c49b950655c,['section_rename'])
```

```
Successful (score):  
None
```

```
Successful (score):  
(4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1e  
a45308b8c49b950655c,['section_rename',  
'section_rename', 'section_rename'])
```

# Wasn't great results but was helpful.

- It wasn't entirely random... since the best action selection resulted in the same set of samples across all testing to evade with similar actions as the Boltzmann Q Policy.
- The model did learn something...
- The inconsistency between the number of same actions for the same samples strongly suggests that the probability distribution in the action space: eg. selecting a benign section name from existing names affects the Malware's ability to evade.



# Things I'd change (Again)

- Change the action space, separate any actions that randomly select a benign feature from a pool into separate actions.

Eg. “rename\_section” would be partitioned into actions:

“rename\_section\_benign1”, “rename\_section\_benign2”....etc.

This will allow the policy to select the best action in rename\_section and create its own probability distribution for the actions.

# Takeaways/Future Ideas

- Was able to create 953 modified Ransomwares from at most 5 samples with unique sha256 signatures.
- An ensemble of models where their generated samples are unioned, say the random select model and the trained model, then tested against the Classifier will result in this particular selection of Ransomware, 100% of the classified malicious samples being capable of fooling the Classifier.
- Take the model and turn it into a generator for a Malware Classifier to use to train on for data augmentation.

# Execute the Project!

- Samples can be taken from:

<https://github.com/ytisf/theZoo>

- Jupyter Notebook:

<https://colab.research.google.com/drive/1I-dXUroohAwgfmHijq03yILvC-DHnu8N>

Remember if you plan to resume training at a later date and store it on Google Drive, to encrypt the compressed folder.

# Citations

- Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, Phil Roth, "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning", in ArXiv e-prints. Jan. 2018.